PPDG Site AAA Issues List

The PPDG Site AAA project has two parallel working documents. This first is an issues list to capture the concerns with function and operation of the GRID tools currently under PPDG review. Some of these issues will have various resolutions with varying requirements. Some will have common requirements and/or resolution.
The second document in the set is a requirements list that all acceptable software must meet. It is expected that not all issues result in requirements, however all requirements should have a corresponding issues discussion.

1. Identity and Registration

Each Grid entity (a person, a machine, or a process) that needs to engage in a Grid communication must have one (or more) Grid identities. The Grid Identity identifies the party in the Grid communications and must suitably resolve ambiguity. It may be the case that more than one entity can assert a Grid Identity, but a Grid Identity must refer to a well defined set of entities.

1. Certificate Authorities

The primary purpose of Certificate Authorities is to provide certificates which define a Grid Identity. The identity consists of at least two parts – a unique name and the public x509 key -- bound together in a certificate. Sites expect that Certificate Authorities operate a service that ensures that

a. certificates are unique,
b. CA infrastructure is operated such that unauthorized certificates aren't created,

c. the certificate is delivered to the appropriate party.

2. Registration Authorities

Identifying the appropriate party is often separated from the function of generating the certificate. The Registration Authority is the CA's agent who matches a person with a request. The sites rely on the Certificate Authorities to ensure that the Registration Authorities:

1. determine that the person requesting the creation (or revocation) of an identity is the/an appropriate person to do so.
2. determine that the person requesting the creation (or revocation) of an identity meets the qualifications required by policy.
3. gathers all required information and verifies that it is correct and/or comes from a trusted source
4. maintains the ability to contact the individual granted an identity in case of incident handling.

3. Name Constraints

In order to ensure that a Grid Identity is unique, namespaces for each CA are defined and namespace limitations need to be enforced. That is, policy in the system must deny certificates issued by CA X that are not within the namespace of the CA (NASA CA can't issue certificates in the name of DOE Science Grids, and visa versa). Currently this is enforced by each Grid Resource maintaining a signing policy file for each CA that is trusted. This is a maintenance burden and investigation of use of x509 name constraints may lead to efficiencies here (as well as remove a

source of configuration trouble). The use of name constraints is not widespread in the SSL community, so this will have to be evaluated for optimal efficiency. The responsibility for determining that there are no overlapping namespaces is not defined at this time. The current default is that each Resource Provider must determine this for themselves as part of the decision to accept credentials from a particular CA.

4. VO Membership Registration

Sites, as resource providers, have a common requirement that users of their resources agree to an acceptable use policy. Since the CAs do not usually know which resources might be used, they are not in a position to act as agents for the sites. The VOs are expected to act as agents for the sites in this regard. To do so, sites will rely on VOs to have a registration process that collects the necessary information, verifies it, and ensures that the the user accepts the appropriate Acceptable Use Policies. This process is quite similar to that run by the RA's above, but in general, they do not have access to each others data. It is not clear whether this is in fact desirable, but the tools and methods should be common.

2. Authentication

Authentication refers to the process of determining that the entity asserting an identity is the intended one(s). Authentication credentials are the data used to prove authentication. Authentication is typically of 3 different types of entities, that have different assumptions and natural methods: Users, Hosts and Services. The GSI/PKI authentication methods for Hosts and Services are quite similar in principle (if not revocation usage) to those in

common use now. The discussion below concentrates on User Authentication.

Implicit in the discussion of authentication is the question of incident handling. All authentication systems can be defeated (some more easily than others) and all rely on some parties to keep secrets. The question of who is responsible to do what when an authentication is challenged (for example, when the request made is deemed to be harmful) depends on the authentication system used and has not been explored in detail in this project. Discussions between representatives of all parties (users, Identity providers, Resource providers, Virtual Organizations, and Resource Owners) need to be held to reach the best compromise in convenience, efficiency and risk acceptance.

1. Interactive User Authentication:

   Under the current Globus Toolkit infrastructure, a user authenticated request is commonly generated today in one of two ways:

   1. The most common situation is that a user maintains a private key in an encrypted form on their local machine. When they want to compute "on the Grid", they decrypt this key (by providing a passphrase) and use it to generate a temporary X.509 proxy credential, which is then used to authenticate subsequent requests to remote resources.
   2. An alternative strategy, used at some sites, is for the user to authenticate to an online Proxy Generation Service and create the proxy credentials. The Kerberos Certificate Authority (KCA) provided by the NSF Middleware Initiative is one example of such a service.

3. A third method uses "smartcard" solutions (virtual or physical) which escrow the individual's private key and effectively act as individual proxy generators. Since the smartcards do not export the long-term private key and are much more resistant to attack than desktop systems, they share many of the same features as the on-line proxy generator service. For purposes of this discussion, they are treated as equivalents to method #2. MyProxy (from NCSA) and Virtual Smart Card (from SLAC) are examples of this method.

NB: We consider proxy repositories (e.g. MyProxy) to be distinct from proxy generation services (e.g. KX509) because in the former, the user may still retain the ability to make a proxy from (another copy of) the long-term credential. In the second, the user does not have that ability.

In approach 1.1.1 above, two forms of credential are at risk:

a. The user private key exists on a user-owned file, in encrypted form. So there are presumably several possible risks to be concerned about:
    i. the user might make that file world readable (and surely some users will do so), in an environment in which other users have access to the relevant file system
    ii. storage system security is such that read access to the private key file is vulnerable to capture (e.g.. network sniff of file system transfer, etc.)

iii. the user might choose a pass phrase that is easily "broken" by someone who gains access to the file system

b. The proxy credential private key exists in a user-only-readable file, in *unencrypted* form. This key is vulnerable to the same exposure risks noted above. However, the value of this key is time-limited and that lifetime cannot be altered by the possessor of the key. Therefore the vulnerability introduced here is similar to that of many other successfully deployed systems (AFS, Kerberos, etc.)

In the case of proxy generators and smartcards, usually, only the second credential is at risk to user/system misconfiguration. The first credential is at risk to theft/misuse via two primarily means:

c. the user who exposes the access secret needed to generate the proxy (e.g.. password written on desktop, etc.)
d. misconfigured or vulnerable proxy generation servers might allow unauthorized users to access/create proxies.

There is considerable discussion on what protection measures are necessary for the short-lived proxy credential as well as services that create them. It's clear that the lifetime of the proxy is a critical parameter in those discussions.

2. Unattended User Authentication:

There is a hybrid case (unattended user jobs) that has characteristics of both a user and service. The most frequent manifestation of this usage case are batch jobs and cron jobs. One can think of cron as a very simplistic batch system. In such a case, some service (the batch system, cron, etc.) is receiving a request to perform some task on behalf of the user. There are two authentications needed, which may in principal be widely separated in time. First the authentication of the request for a command to be run and second the authentications needed by the command at time of execution. The first can, in almost all cases, use the normal user authentication methods described above and is pseudo-realtime. The second typically wants to grant the user's authorizations (or a dynamic, usually difficult to define subset) to the command. There are three general approaches:

1. The executed command authenticates as the user and requests made are indistinguishable from those made interactively by the user.
2. The command authenticates as an identity algorithmically derived from (but separate from) the user. Commands are issued as that derived identity.
3. The command authenticates as the "batch" service.

Whichever approach is taken, the operator of the batch service will necessarily have the ability to authenticate as the identity used for the lifetime of the authentication "secret". Depending on the skill of its administration, attackers of the batch system can gain that ability as well.

Approach 1 exposes the users (sole ?) identity to the full risks of unattended operation and allows for no distinguishing action in event of compromise/failure.

Approach 2 requires the maintenance of multiple identities per person (though they may be automatically associated with the primary identity) and for specific inclusion of those separate identities in the resource access control lists.

Approach 3 presumes the resources to be accessed by the job are either a) fully available to (any user of) the batch service or b) managed by a service that can carry on a trusted authorization based on the user identity as authenticated by the batch service.

Different elements of the Grid architecture are required to implement services (and enforce policy) depending on which approach is taken. It is not clear whether all approaches are mutually compatible or which are operationally preferred. This is an area in need of concentrated development.

3. Revocation of Authentication:

All authentication relies on some secret (password, private key file, hardware token, etc.) that can be compromised and used by unauthorized persons. In this regard, user, host, and service authentication share a common concern.

Determining whether authentication secrets (and which) have been compromised and preventing (further) exploit of the secret for unauthorized used is a core part of incident handling. The questions of what constitutes a compromise of a private key and who needs to do what in cases of private keys believed to be

compromised have yet to be resolved. The scope of this question is also beyond the scope of just sites (Resource Providers) and includes Identity Providers, VOs, users and operations managers.

In the event of a compromise, that authentication ability must be revocable on a timescale appropriate to the compromise. For example, the timescale for the need of revocation of a stolen private key file is a function of the strength of its passphrase encryption. If there is none (or the passphrase is also stolen), then the timescale needed is immediate. If it is a still secure, 20 character, random passphrase, the timescale is millennia. A standard operational assumption is that revocation needs to happen within ~24 hours. Authorization restriction methods are presumed to handle reaction times shorter than that.

Every authentication process needs to invoke tests to determine if the authentication "secret" is a) correct and b) has not been revoked.

In the case of compromise, one does not usually want to invalidate the identity, but rather the authentication secret (replacing it and invalidating the earlier one). In most authentication systems widely deployed today, the system queried to determine correctness of the authentication secret is the same one that determines its validity. Thus updating the authentication system is a simple matter of updating the copy (or hash) of the secret held by the authentication server. (For example, one sets a new password with: the local password file, the NIS password file, the KDC (AFS, W2K, KRB5)). With PKI those two functions are split. One can test the correctness of the authentication secret (possession of the private key) directly. This is the basis of the claimed benefit of PKI that no critical central

service is required. However, there is no way to determine by inspection whether a authentication secret which was once valid, is still valid. To do so one must consult an independent authority, which introduces a critical service back into the picture. Furthermore, there is no way, once the private key corresponding to certificate has been compromised, to "fix" the certificate with a new private key. It must be abandoned -- permanently and universally.

The Grid Identity is the certificate generated by a trusted CA. There may be multiple certificates (valid, expired, and revoked) issued for any one individual. To determine which certificate/private key pairs are valid one has to consult the certificate issuer. Since a validity decision is time dependent, this check must be done for every authentication.

(Alternately, one could refuse to revoke authentication secrets and push this responsibility onto authorization. Regardless, there has to somewhere be a reliable assertion that the authentication secret(s) is(are) not known to be compromised.)

The current Globus method of determining if a certificate is still valid is to presume success and examine a Certificate Revocation List (CRL), if available. (The presumption of success and fail open decision means the system is vulnerable to an attacker who can block access to the CRL.) It requires each relying party to have access to a CRL (or an on-line lookup) for each CA in every certificate chain presented. The maximum allowed age of the CRL is the maximum tolerated latency for revoking certificates. (i.e.. to have a 1 day response, one must get new CRLs every day or use on-line lookups.)

The CRL is unique to each signing party. Thus in a chain of certificates, not only must the signature be checked, but also that that the signing CA's CRL does not list the certificate signed as invalid. An example is probably in order (since my brain hurts at that text ;-). If CA A generates a certificate for CA B who generates a certificate for user C, then to determine if the authentication for user C is valid, one must:

e. Check that the proof of possession of private key for C (using the certificate for user C) succeeds.
f. Check that CA B's signature of C's certificate is valid (using the certificate for CA B (generated by CA A)).
g. Check that C's certificate is still valid (i.e. not expired, not on the CRL for CA B, etc.)
h. Check that CA B is allowed to generate C's certificate. (i.e. name constraints obeyed)
i. Check that CA A's signature of B's certificate is valid (using the certificate for CA A stored on the system).
j. Check that CA A's certificate is still valid (i.e. not expired)
k. Check that CA A is allowed to generate B's certificate.

All this must succeed to have reliable authentication.

3. Authorization

Authorization is the process of determining whether the identity may be granted their request. An authorization token is some data, the possession of which, perhaps in conjunction with some other authentication or identity information (most commonly the private key corresponding to the Grid Identity involved) allows an entity to prove authorization. Authorization tokens are not handled in any standard fashion in the Grid. If they are deemed to be useful,

and there is much discussion on this matter, then handling them will require standardization efforts.

The authorization decision process consists of a number of questions. Discussions of authorization engender a number of questions. The two sets are mixed below.

1. Who are the necessary authorizing parties ?

   This may in fact, be a complex question and require a syntax for expressing requirements in general. However, our discussions seem to keep coming back to a three tier system: resource manager, resource owner (site), VO. If each of these allows arbitrary complexity, then it seems reasonable that one could cover the required space with these three entities. However, since it's more than 2, dealing with the case of arbitrary number of authorizing parties may be an easy extension of this minimum.

   A possible solution would be to use a PAM-like framework for authorization decisions at the Resource level. In this model, the Resource Managers would be responsible for structuring the authorization logic appropriately for their resource. This would involve a negotiation with the parties to which they provide service and result in a decision tree using decision modules provided by the authorizing parties. As an example, a general Compute Element at Fermilab, would have a decision tree something like this:

   1. Check FNAL site authorization
   2. Loop over VO membership attributes until pass
      - If CDF_member=true

        - check CDF authorization (is this a hierarchy itself ?)
        - check Resource CDF authorization
    - If D0_member=true
        - check D0 authorization
        - check Resource D0 authorization
    - If CMS_member=true
        - check CMS authorization
        - check Resource CMS authorization
    - else
        - check Default authorization
        - check Resource Default authorization

    end loop

3. Fail if no authz check passes.

This would imply that the VO membership information is available with the request (e.g. in a attribute in the proxy certificate). Alternately, one could have the membership checks done in realtime (at the expense of another critical path service).

2. **What data may authorization decisions require ?**

In principal, authorization decisions can be based on any arbitrary data the authorizing party chooses. The presumption that the information presented in the SSL (or GSI) connection authentication is sufficient is, in general, false. Already within the sample of the 5 labs participating in this study, we are seeing instances where the GSI information is insufficient to meet site requirements except in very restrictive configurations. Furthermore, applying different authorization requirements based on the request

being made is not accommodated. To allow Grid Resources to have autonomy in their authorization, the request authorization interface has to be generalized. There are at least 3 ways this could be done:

1. Have the resource advertise the information needed for authorization and have the requestor present this information with the GSI credential in a standard fashion.
2. Have the resource negotiate authorization methods (and information) with the requestor (ala SASL).
3. Have the resource fork a separate authorization process to obtain the needed authorization information from the user (or the user's agent).

The first option means that a standardized method of encoding authorization information in GSI proxies has to be developed. This is the method being pursued by the EDG and CAS projects. It further assumes that all authorization tokens can be presented securely by the client without a interactive response from the resource. This makes challenge/response methods difficult to deal with (but perhaps the presumption that a Grid Resource cannot have interactive response back to the submitting user is a practical one). It requires the requestor to appropriately construct the proxy based on the service being requested..

The second option is more forgiving of requestor preparation, but requires a more complex protocol than the current GSI. A framework like SASL that allows for client software to be enhanced with new methods by addition of a (system) library and for servers to present a list of acceptable methods would be appropriate here and a useful way to avoid frequent redistributions of clients to permit new methods (or fix old ones).

The third would have to be able to determine the reverse mapping from Grid Identity to the appropriate callback location/method. While this may be a viable option in the OGSA, it is not being investigated to our knowledge.

When is authorization checked ?

The current model is that authorization to receive a service is checked (only) at the time of the request. Since there are requests down a hierarchy of Grid Services to the lowest level Grid interface, functionally this means that authorization is checked by every Grid Service at its own discretion. Once a service request has been granted is there reason to force rechecks of the authorization ? This seems best to be an item best left to the Grid Service provider. They could implement periodic rechecks at their discretion, but there seems no systematic reason to insist on it.

3. How is authorization revoked ?

The initial answer seems to be for authorized actions to be atomic and have no revocation method. For the purposes of handling tokens, this may be acceptable. However, it must be possible for an authorizer of an atomic action to kill the request. Consider the case of the user who submits 10 copies of a job, 9 long and one short

test job. If examination of the test job indicates the code has a bug, the user may well want to kill the 9 long jobs even/especially if they are currently running. Is this to be accomplished by revoking the authorization of the existing jobs or by issuing a second request to every grid resource used asking to abort the previous one ?

4. **May authorization be delegated ?**

   It may be acceptable for some requests to allow the requestor to delegate the authorization to perform the request. In this scenario, the holder of some authorization token would create a delegated authorization token specifying that it authorized the second entity to use it's initial authorization. This would have to be checked for acceptability by the issuer of the delegated token and the resource accepting the token. Is this useful ?

5. **Is authorization information private ?**

   Particularly in the case where authorization tokens are presented with the GSI certificate, is there reason to obscure the authorization information. It would seem that exposure of the detailed list of rights (authorization tokens) a requestor might have would create a way of targeting privileged identities. Merely exposing the authorizing entity may not reveal too much information (though for authorizing entities dedicated to high value credentials, this would be sufficient.

   Since, in general, the requestor does not know the identity of the relying party, how would this be done ?

4. Auditing

The issues with auditing are not well understood in the context of HEP collaborative science. It is clear that the interest in “chargeback” style account

will be greater than in the past since the information will not only be used for determining if resources are being fairly shared, but also whether production commitments are being met. Auditing will have key roles in resource allocation, incident handling, quality assurance, and operations management in ways that will be different than in previous paradigms. These issues have come up in our discussions, but it is quite likely there are significant issues yet to resolve. In our discussions, auditing is distinguished from accounting in that auditing is the system design aspect that ensures that actions can be traced on demand within the agreed limits. Accounting is the process of summarizing actions by (various) organization. It is our opinion that both will be more important than before to audiences beyond the resource operator and that auditing is the more urgent need and more intimately tied to authentication and authorization.

1. **Who is responsible for keeping what usage information ?**

   Every Grid entity that provides a service to another may need to be audited by the relying service for troubleshooting. Defining who needs to keep what, for how long, and to whom it may/should/must be made available (and how) are important operational design issues.

2. **Is reverse mapping from local identity to invoking Grid identity available for appropriate accounting ?** particularly in case of mapping onto shared or transient local accounts ?

   Since all Grid transactions will use Grid Identities and most currently existing accounting and audit methods use local identities, it needs to be made clear for what actions a Grid Resource must provide the reverse translation (and to whom it should report the information).

3. **Who defines what level of accounting is required and how ?**

We believe this is largely a VO matter, but that sites will have internal requirements for their management control, for debugging purposes. Sites will also have requirements for standard interfaces for reporting this information to minimize the need for multiple implementations.

PPDG Site AAA Mailing List

Last modified: Wed Jan 22, 2003